Set Smart Contract: Review

Mikhail Vladimirov and Dmitry Khovratovich

04th June 2019

This document describes the issues, which were found in the Set smart contract during the code review performed by ABDK Consulting.

1. Introduction

We were asked to review a set of contracts in a private repo aling with some documentation.

2. DailyPriceFeed

In this section we describe issues related to the smart contract defined in the DailyPriceFeed.sol

2.1 Moderate Flaw

Line 114: the `medianizerInstance.read()` allows medianizer to perform recursive call attack on this contract. To prevent it, update `lasUpdateAt` storage variable before calling medianizer.

2.2 Documentation and Readability Issues

This section lists documentation issues, which were found in the smart contract, and the cases where the code is correct, but too involved and/or complicated to be verified or analyzed.

1. Line 39: `24 hours` instead of `86400` would be more readable.
2. Line 64, 153: there should be `IMedian` instead of `address`.
3. Line 102: the period between calls is greater or equal to 24 hours, so average period on long run will be more than 24 hours.
4. Line 107: to ensure daily updates, one can just check that the next astronomical day has started, no matter more or less 24 hours have passed.

2.3 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 34: the `LinkedListLibrary` is structured to be used via "using LinkedListLibrary for LinkedListLibrary.LinkedList", not to be inherited from.
2. Line 43: the storage variable `medianizerAddress` always has the same value as medianizerInstance.
3. Line 45: the string `dataDescription` is never used in the contract. Consider logging it and storing the hash only, or just logging it.
4. Line 47: the prefix `LinkedListLibrary.` is not needed (the smart contract inherits from library).
5. Line 95: in order to make poking every 24 hours possible this should be "lastUpdatedAt.add (24 hours)".
6. Line 128: query will revert in case when not enough days are collected or if more days asked than stored (even though we had collected them earlier).

3. LinkedListLibrary

In this section we describe issues related to the smart contract defined in the LinkedListLibrary.sol.

3.1 Moderate Flaws

This section lists moderate flaws found in the smart contract.

Line 60:the `push(_initialValue)` does not guarantee saving at position zero, because `dataArray` may be not empty.

Line 61: in addition to the previous comment, the value at this line (`0`),may he incorrect.

Line 107: the `push(_addedValue)` is not guaranteed to save value at `newStoreIndex`.

Line 110: the value `newNodeIndex` may be incorrect (according previous comment).

3.2 Documentation and Readability Issues

This section lists documentation issues, which were found in the smart contract, and the cases where the code is correct, but too involved and/or complicated to be verified or analyzed.

1. Line 35: the fields in the `LinkedList` structure should be documented.
2. Line 71: the name of the function `editList` is confusing. Common name for insert-or-update functions is `put`.

3.3 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 29: the data structure of the `LinkedListLibrary` contract looks like overkill for moving average calculation task. Common solution is to used fixed

size array as ring buffer and single accumulator. New values are added to the accumulator and removed values are subtracted from it.

2. Line 37: storing next index to be updated instead of last updated index is more common and would make code simpler.
3. Line 38,39: the fixed size array would be cheaper than dynamic one.
4. Line 60: the method `dataArray` should be prevented from calling twice or the array must be deleted before push.
5. Line 104: links `[x]` equals to `x-1` for all used slots except for `slot #0`. And `links[0]` is always `dataSizeLimit-1`. There is no reason to store links array at all, because its values may easily be calculated.
6. Line 120: the method is overcomplicated. Could be implemented in one line: `_self.dataArray [_self.lastUpdateIndex = _self.lastUpdateIndex.add(1) % _self.dataSizeLimit] = _addedValue`
7. Line 157: the real consumers need sum of values rather than values themselves.

3.4 Other Issues

This section lists stylistic and other minor issues found in the smart contract.

Line 175: the operation in this line just wastes gas in the last iteration.

4. MovingAverageOracle

In this section we describe issues related to the smart contract defined in the MovingAverageOracle.sol

4.1 Arithmetic Overflow Issues

This section lists issues of smart contract related to the arithmetic overflows.

Line 83: while average of uint256 values always fits into uint256, overflow is possible in this line. This is bad practice to allow overflow in intermediate calculations even though the final result fits into target type.

4.2 Suboptimal Code

This section lists suboptimal code patterns, which were found in the smart contract.

1. Line 36: the storage `priceFeedAddress` variable always has the same value as `priceFeedInterface`.
2. Line 52: there should be `IDailyPriceFeed` instead of `address`.
3. Line 82: storing individual prices and summing them up every time is subeffective. Better solution would be to store cumulative sums of prices. I.e. lets p_i be price #i. Instead of storing p_i, you should store s_i = p_1 + p_2 + … p_i. Then to calculate sum of prices in arbitrary range [a..b] you need to

just subtract one cumulative sum from another: p_a + ... + p_b = s_b - s_(a-1), assuming s_0 = 0.

4.3 Other Issues

This section lists stylistic and other minor issues found in the smart contract.

1. Line 75: prices are `uint256` but their average is `bytes32`.

5. Supporting Documentation

In this section we describe issues found in the Set Strategies Introduction.

1. According to the document “Simple Moving averages (SMA) is a widely used indicator in technical analysis and is simply the average of the price of an asset over a defined number of time periods”.This not provided case then price moves within a period.
2. The **poke()** description: “Requires 24 hours have passed”. In this case average interval between pokes will be greater than 24 hours.

Our Recommendations

Based on our findings, we recommend the following:

1. Fix the moderate issues.
2. Fix arithmetic overflow issues.
3. Refactor the code to remove suboptimal parts.
4. Fix the documentation, readability and other (minor) issues.